UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic
ECE 666

Part 4-B
Floating-Point Arithmetic - II

Israel Koren

The IEEE Floating-Point Standard

- Four formats for floating-point numbers
- First two:
  - basic single-precision 32-bit format and
  - double-precision 64-bit format
- Other two - extended formats for intermediate results
- Single extended format - at least 44 bits
- Double extended format - at least 80 bits
- Higher precision and range than corresponding 32- and 64-bit formats

Single-Precision Format

- Most important objective - precision of representation
- Base 2 allows a hidden bit - similar to DEC format
- Exponent field of length 8 bits for a reasonable range

S	8 bits - biased exponent E	23 bits - unsigned fraction f

- 256 combinations of 8 bits in exponent field
  - * E=0 reserved for zero (with fraction f=0) and denormalized numbers (with fraction $f \neq 0$)
  - * E=255 reserved for $\pm\infty$ (with fraction f=0) and NaN (with fraction $f \neq 0$)
- For 1 < E < 254 -

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

IEEE vs. DEC

- Exponent bias - 127 instead of $2^{e-1} = 2^7 = 128$
- Larger maximum value of true exponent - 254-127=127 instead of 254-128=126 - larger range
- Similar effect - significand of 1.f instead of 0.1f -
- Largest and smallest positive numbers -

$$F^+_{max} = (2 - 2^{-23}) \cdot 2^{254-127} = (1 - 2^{-24}) \cdot 2^{128}$$

$$F^+_{min} = 1.0 \cdot 2^{1-127} = 2^{-126}$$

- instead of

$$F^+_{max} = (1 - 2^{-24}) \cdot 2^{127} \text{ and } F^+_{min} = 2^{-128}$$

- Exponent bias and significand range selected to allow reciprocal of all normalized numbers (in particular, F^+_{min}) to be represented without overflow - not true in DEC format

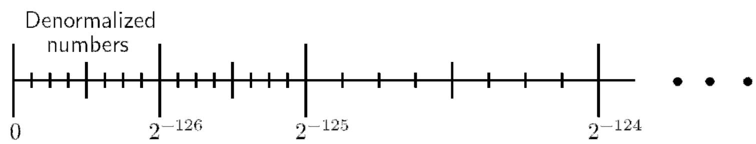
Special Values in IEEE Format

	$f=0$	$f \neq 0$
$E=0$	0	Denormalized
$E=255$	$\pm\infty$	NaN

- $\pm\infty$ - represented by $f=0$, $E=255$, $S=0,1$ - must obey all mathematical conventions: $F+\infty=\infty$, $F/\infty=0$
- Denormalized numbers - represented by $E=0$ - values smaller than smallest normalized number - lowering probability of exponent underflow
- $F=(-1)^S \cdot 0.f \cdot 2^{-126}$
- Or - $F=(-1)^S \cdot 0.f \cdot 2^{1-127}$ - same bias as normalized numbers

Denormalized Numbers

- No hidden bit - significands not normalized
- Exponent - -126 selected instead of $0-127=-127$ - smallest normalized number is $F^+_{min}= 1 \cdot 2^{-126}$
- Smallest representable number is $2^{-23} \cdot 2^{-126} = 2^{-149}$ instead of 2^{-126} - gradual (or graceful) underflow
- Does not eliminate underflow - but reduces gap between smallest representable number and zero; 2^{-149}= distance between any two consecutive denormalized numbers = distance between two consecutive normalized numbers with smallest exponent $1-127=-126$



Denormals & Extended formats

- Denormalized numbers not included in all designs of arithmetic units that follow the IEEE standard
  - Their handling is different requiring a more complex design and longer execution time
  - Even designs that implement them allow programmers to avoid their use if faster execution is desired
- The single-extended format for intermediate results within evaluation of complex functions like transcendental and powers
- Extends exponent from 8 to 11 bits and significand from 23+1 to 32 or more bits (no hidden bit)
  - Total length is at least 1+11+32=44 bits

NaN (E=255)

- $f \neq 0$ - large number of values
  - Two kinds - signaling (or trapping), and quiet (nontrapping) - differentiated by most significant bits of fraction - remaining bits contain system-dependent information
  - Example of a signaling NaN - uninitialized variable
  - It sets Invalid operation exception flag when arithmetic operation on this NaN is attempted ; Quiet NaN - does not
  - Turns into quiet NaN when used as operand if Invalid operation trap is disabled (avoid setting Invalid Op flag later)
  - Quiet NaN produced when invalid operation ($0 \cdot \infty$) attempted - this operation had already set the Invalid Op flag once. Fraction field may contain a pointer to offending code line
  - Quiet NaN, as operand will produce quiet NaN result and not set exception. For example, NaN+5=NaN. If both operands quiet NaNs, result is the NaN with smallest significand

Double-Precision Format

- Main consideration - range; exponent field - 11 bits

S	11 bits - biased exponent E	52 bits - unsigned fraction f

- E=0,2047 reserved for same purposes as in single-precision format
- For $1 \leq E \leq 2046$ - $$F = (-1)^S \, 1.f \; 2^{E-1023}$$
- Double extended format - exponent field - 15 bits, significand field - 64 or more bits (no hidden bit), total number of bits - at least 1+15+64=80

	Single	Double
Word length	32 bits	64 bits
Fraction + hidden bit	23 + 1 bits	52 + 1 bits
Exponent	8 bits	11 bits
Bias	127	1023
Approximate range	$2^{128} \approx 3.8 \cdot 10^{38}$	$2^{1024} \approx 9 \cdot 10^{307}$
Smallest normalized number	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Approximate resolution	$2^{-23} \approx 10^{-7}$	$2^{-52} \approx 10^{-15}$

Round-off Schemes

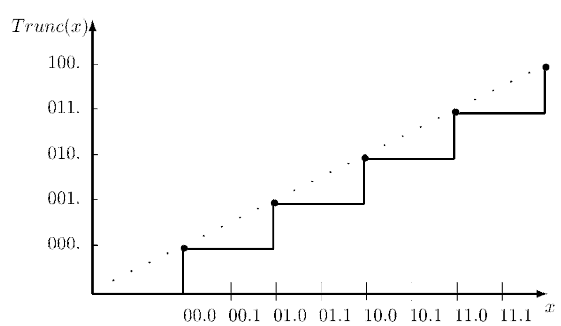
- Accuracy of results in floating-point arithmetic is limited even if intermediate results are accurate
- Number of computed digits may exceed total number of digits allowed by format - extra digits must be disposed of before storing
- Example - multiplying two significands of length m - product of length 2m - must be rounded off to m digits
- Considerations when selecting a round-off scheme -
  - * Accuracy of results (numerical considerations)
  - * Cost of implementation and speed (machine considerations)

Requirements for Rounding

- x,y - real numbers; Fl - set of machine representations in a given floating-point format; $Fl(x)$ - machine representation of x
- Conditions for rounding:
  - $Fl(x) \le Fl(y)$ for $x \le y$
  - If $x \in Fl$ - $Fl(x)=x$
  - If $F1$, $F2$ consecutive in Fl and $F1 \le x \le F2$, then either $Fl(x)=F1$ or $Fl(x)=F2$
- d - number of extra digits kept in arithmetic unit (in addition to m significand digits) before rounding
- Assumption - radix point between m most significant digits (of significand) and d extra digits
- Example - Rounding 2.99_{10} into an integer

Truncation (Chopping)

- d extra digits removed - no change in m remaining digits - rounding towards zero
- For $F1 \le x \le F2$ - $Trunc(x)$ results in $F1$ ($Trunc(2.99)=2$)
- Fast method - no extra hardware
- Poor numerical performance - Error up to ulp
- $Trunc(x)$ lies entirely below ideal dotted line (infinite precision)



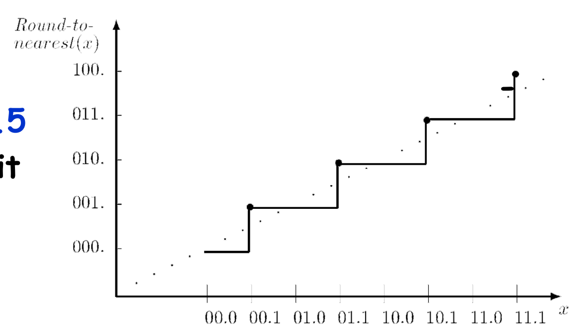
Rounding Bias

- Rounding bias - measures tendency of a round-off scheme towards errors of a particular sign
- Ideally - scheme is unbiased or has a small bias
- Truncation has a negative bias
- Definition - Error=Trunc(x)-x ; for a given d - bias is average error for a set of 2^d consecutive numbers with a uniform distribution
- Example - Truncation, d=2
- X is any significand of length m
- Sum of errors for all $2^d=4$ consecutive numbers=-3/2
- Bias=average error=-3/8

Number	$Trunc(x)$	Error
$X.00$	X	0
$X.01$	X	$-1/4$
$X.10$	X	$-1/2$
$X.11$	X	$-3/4$

Round to Nearest Scheme

- $F1 \le x \le F2$ - Round(x)=nearest to x out of F1,F2 - used in many arithmetic units
- Obtained by adding 0.1_2 (half a ulp) to x and retaining the integer (chopping fraction)
- Example - x=2.99 - adding 0.5 and chopping off fractional part of 3.49 results in 3
- Maximum error - x=2.50 - 2.50+0.50=3.00 result=3, error=0.5
- A single extra digit (d=1) is sufficient



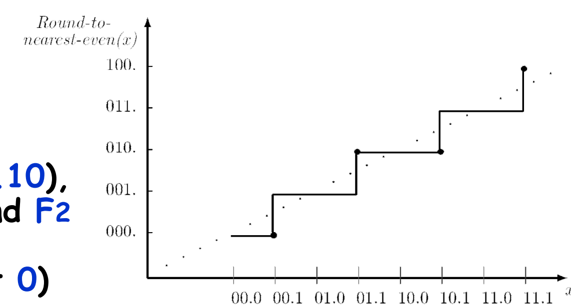
Bias of Round to Nearest

- **Round(x)** - nearly symmetric around ideal line - better than truncation
- Slight positive bias - due to round up of X.10
- d=2 :

Number	*Round-to-nearest(x)*	Error
$X.00$	X	0
$X.01$	X	$-1/4$
$X.10$	$X + 1$	$+1/2$
$X.11$	$X + 1$	$+1/4$

- Sum of errors=1/2, bias=1/8, smaller than truncation
- Same sum of errors obtained for d>2 - bias=$1/2 \cdot 2^{-d}$

Round to Nearest Even



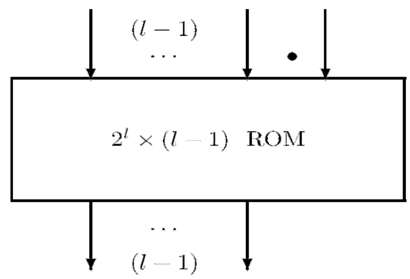
- In case of a tie (X.10), choose out of F_1 and F_2 the even one (with least-significant bit 0)
- Alternately rounding up and down - unbiased
- Round-to-Nearest-Odd - select the one with least-significant bit 1
- d=2 :
- Sum of errors=0
- Bias=0

Number	*Round(x)*	Error	Number	*Round(x)*	Error
$X0.00$	$X0.$	0	$X1.00$	$X1.$	0
$X0.01$	$X0.$	$-1/4$	$X1.01$	$X1.$	$-1/4$
$X0.10$	$X0.$	$-1/2$	$X1.10$	$X1.+1$	$+1/2$
$X0.11$	$X1.$	$+1/4$	$X1.11$	$X1.+1$	$+1/4$

- Mandatory in IEEE floating-point standard

ROM Rounding

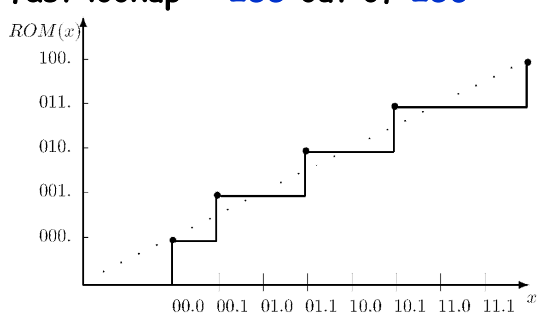
- Disadvantage of round-to-nearest schemes - require a complete add operation - carry propagation across entire significand
- **Suggestion** - use a **ROM** (read-only memory) with look-up table for rounded results
- **Example** - a **ROM** with ℓ address lines - inputs are ℓ-1 (out of m) least significant bits of significand and most significant bit out of d extra bits



ECE666/Koren Part.4b.17

ROM Rounding - Examples

- **ROM** has 2^ℓ rows of ℓ-1 bit each - correct rounding in most cases
- When all ℓ-1 low-order bits of significand are 1's - **ROM** returns all 1's (truncating instead of rounding) avoiding full addition
- **Example** - ℓ=8 - fast lookup - 255 out of 256 cases are properly rounded
- **Example:** ℓ=3



ECE666/Koren Part.4b.18

Bias of ROM Rounding

- Example - $\ell=3$; d=1
- Sum of errors=1
- Bias=1/8

Number	$ROM(x)$	Error	Number	$ROM(x)$	Error
$X00.0$	$X00.$	0	$X10.0$	$X10.$	0
$X00.1$	$X01.$	+1/2	$X10.1$	$X11.$	+1/2
$X01.0$	$X01.$	0	$X11.0$	$X11.$	0
$X01.1$	$X10.$	+1/2	$X11.1$	$X11.$	−1/2

- In general - bias=$1/2[(1/2)^{d}-(1/2)^{\ell-1}]$
- When ℓ is large enough - ROM rounding converges to round-to-nearest - bias converges to $1/2(1/2)^{d}$
- If the round-to-nearest-even modification is adopted - bias of modified ROM rounding converges to zero

Rounding and Interval Arithmetic

- Four rounding modes in IEEE standard
  * Round-to-nearest-even (default)
  * Round toward zero (truncate)
  * Round toward ∞ * Round toward −∞
- Last 2 - useful for Interval Arithmetic
  * Real number a represented by lower and upper bounds a_1 and a_2
  * Arithmetic operations operate on intervals
  * Calculated interval provides estimate on accuracy of computation

$$[a_1, a_2] + [b_1, b_2] = [a_1 + b_1, a_2 + b_2]$$

$$[a_1, a_2] - [b_1, b_2] = [a_1 - b_2, a_2 - b_1]$$

$$[a_1, a_2] \times [b_1, b_2] = [\min\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}, \max\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}]$$

  * Lower bound rounded toward −∞, upper - toward ∞

Guard Digits for Multiply/Divide

- Multiplication has a double-length result - not all extra digits needed for proper rounding
- Similar situation - adding or subtracting two numbers with different exponents
- How many extra digits are needed for rounding and for postnormalization with leading zeros ?
- Division of signed-magnitude fractions - no extra digits - shift right operation may be required
- Multiplying two normalized fractions - at most one shift left needed if $\beta=2$ (k positions if $\beta=2^k$) $\Rightarrow$ one guard digit (radix β) is sufficient for postnormalization
- A second guard digit is needed for round-to-nearest - total of two - *G* (guard) and R (round)
- Exercise - Same for range [1,2) (IEEE standard)

Guard, Round and Sticky digits

- Round-to-nearest-even - indicator whether all additional digits generated in multiply are zero - detect a tie
- Indicator is a single bit - logical OR of all additional bits - sticky bit
- Three bits - *G*, R, S (sticky) - sufficient even for round-to-nearest-even
- Computing S when multiplying does not require generating all least significant bits of product
- Number of trailing zeros in product equals sum of numbers of zeros in multiplier and multiplicand
- Other techniques for computing sticky bit exist

Guard digits for Add/Subtract

- Add/subtract more complicated - especially when final operation (after examining sign bits) is subtract
- Assumption - normalized signed-magnitude fractions

F_1	0.	1	0	0	0	0	0	$\cdot$	16^3
F_2 aligned	0.	0	F	F	F	F	F	$\cdot$	16^3
$F_1 - F_2$	0.	0	0	0	0	0	1	$\cdot$	16^3
Postnormalization	0.	1	0	0	0	0	0	$\cdot$	16^{-2}

- Subtract - for postnormalization all shifted-out digits of subtrahend may need to participate in subtraction
  - * Number of required guard digits = number in significand field - double size of significand adder/subtractor
- If subtrahend shifted more than 1 position to right (pre- alignment) - difference has at most 1 leading zero
- At most one shifted-out digit required for postnormalization

Subtract - Example 1

- Calculating A-B
- Significands of A and B are 12 bits long, base=2, E_A-E_B=2 - requiring a 2-bit shift of subtrahend B in pre-alignment step

A	0.100000101100	00
B aligned	0.001100000001	10
$A - B$	0.010100101010	10
Postnormalization	0.101001010101	

- Same result obtained even if only one guard bit participates in subtraction generating necessary borrow

Subtract - Example 2

- Different if most significant shifted-out bit is 0
- Same two significands - $E_A - E_B = 6 \rightarrow$ B's significand shifted 6 positions

A	0.100000101100	000000
B aligned	0.000000110000	000110
$A-B$	0.011111111011	111010
Postnormalization	0.111111110111	

  - If only one guard bit - 4 least significant bits of result after postnormalization would be 1000 instead of 0111
  - Long sequence of borrows - seems that all additional digits in B needed to generate a borrow
- Possible conclusion : in the worst case - number of digits doubled
- Statement : Enough to distinguish between two cases:
  - (1) All additional bits (not including the guard bit) are 0
  - (2) at least one of the additional bits is 1

Proof of Statement

- All extra digits in A are zeros (not preshifted)
- Resulting three least significant bits in A-B (011 in example 2) are independent of exact position of 1's in extra digits of B
- We only need to know whether a 1 was shifted out or not - sticky bit can be used - if 1 is shifted into it during alignment it will be 1 - otherwise 0 - logical OR of all extra bits of B
- Sticky bit participates in subtraction and generates necessary borrow
- Using G and S -

		G	S
A	0.100000101100	0	0
B aligned	0.000000110000	0	1
$A-B$	0.011111111011	1	1
Postnormalization	0.111111110111		

- G and S sufficient for postnormalization
- In round-to-nearest - an additional accurate bit needed - sticky bit not enough - G,R,S required

Example 3 (E_A-E_B=6)

♦ Correct result

A	0.100000101100	000000
B aligned	0.000000110000	010110
$A-B$	0.011111111011	101010
Postnormalization	0.111111110111	0

Using only G and S

		G	S
A	0.100000101100	0	0
B aligned	0.000000110000	0	1
$A-B$	0.011111111011	1	1
Postnormalization	0.111111110111	1	

♦ Round bit after postnormalization - 0, sticky bit cannot be used for rounding

♦ Using G, R, S

		G	R	S
A	0.100000101100	0	0	0
B aligned	0.000000110000	0	1	1
$A-B$	0.011111111011	1	0	1
Postnormalization	0.111111110111	0		

♦ Correct R=0 available for use in round-to-nearest

♦ For round-to-nearest-even: sticky bit needed to detect a tie available - serves two purposes

Example 4 - No Postnormalization

♦ Rounding requires a round bit and a sticky bit

♦ For round-to-nearest-even

* original G can be an R bit
* original R and S ORed to generate a new sticky bit S

♦ E_A-E_B=6

A	0.100001010100			
B	0.110000010001			
		G	R	S
A	0.100001010100	0	0	0
B aligned	0.000000110000	0	1	1
$A-B$	0.100000100011	1	0	1
		R	S	
Before rounding	0.100000100011	1	1	
After round-to-nearest	0.100000100100			

Adding ulp in rounding

- If R=0 no rounding required - sticky bit indicates whether final result is exact/inexact (S=0/1)
- If R=1 operation in round-to-nearest-even depends on S and least-significant bit (L) of result
- If S=1 rounding must be performed by adding ulp
- If S=0 - tie case, only if L=1 rounding necessary
- Summary - round-to-nearest-even requires adding ulp to significand if $RS + R\bar{S}L = R(S + L) = 1$
- Adding ulp may be needed for directed roundings
- Example: in round toward +∞, ulp must be added if result is positive and either R or S equals 1
- Similarly - in round toward - ∞ when result negative and R+S=1

IEEE Format Rounding Rules

LSB	R	S	Operation	$\overline{Error}$
0	0	0	$+0$	0
0	0	1	$+0$	$-0.25\ ulp$
0	1	0	$+0$	$-0.50\ ulp$
0	1	1	$+0.5\ ulp$	$+0.25\ ulp$
1	0	0	$+0$	0
1	0	1	$+0$	$-0.25\ ulp$
1	1	0	$+0.5\ ulp$	$+0.50\ ulp$
1	1	1	$+0.5\ ulp$	$+0.25\ ulp$
			Total	0

(a) Round-to-nearest-even scheme

R	S	Operation	$\overline{Error}$
0	0	$+0$	0
0	1	$+0$	$-0.25\ ulp$
1	0	$+0$	$-0.50\ ulp$
1	1	$+0$	$-0.75\ ulp$
		Total	$-0.375\ ulp$

(b) Round-to-zero scheme

Sign	R	S	Operation
+	0	0	$+0$
+	0	1	$+1\ ulp$
+	1	0	$+1\ ulp$
+	1	1	$+1\ ulp$
−	0	0	$+0$
−	0	1	$+0$
−	1	0	$+0$
−	1	1	$+0$

(c) Round-to-plus-infinity scheme

Sign	R	S	Operation
−	0	0	$+0$
−	0	1	$+1\ ulp$
−	1	0	$+1\ ulp$
−	1	1	$+1\ ulp$
+	0	0	$+0$
+	0	1	$+0$
+	1	0	$+0$
+	1	1	$+0$

(d) Round-to-minus-infinity scheme

Adding ulp in rounding

- Adding ulp after significands were added increases execution time of add/subtract
- Can be avoided - all three guard bits are known before significands added
- Adding 1 to L can be done at the same time that significands are added
- Exact position of L is not known yet, since a postnormalization may be required
- However, it has only two possible positions and two adders can be used in parallel
- Can also be achieved using one adder