



UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Digital Computer Arithmetic

ECE 666

Part 4-A
Floating-Point Arithmetic

Israel Koren
Spring 2008

ECE666/Koren Part.4a.1

Copyright 2008 Koren

Preliminaries - Representation

- ◆ Floating-point numbers - provide a dynamic range of representable real numbers without having to scale the operands
- ◆ Representation - similar to scientific notation
- ◆ Two parts - significand (or mantissa) M and exponent (or characteristic) E
- ◆ The floating-point number F represented by the pair (M, E) has the value -

$$F = Mb^E \quad (b - \text{base of exponent})$$

- ◆ Base - common to all numbers in a given system
- implied - not included in the representation of a floating point number

ECE666/Koren Part.4a.2

Copyright 2008 Koren

Preliminaries - Precision

- ◆ n bits partitioned into two parts - significand M and exponent E
- ◆ n bits - 2^n different values
- ◆ Range between smallest and largest representable values increases \Rightarrow distance between any two consecutive values increases
 - * Floating-point numbers sparser than fixed-point numbers - lower precision
- ◆ Real number between two consecutive floating-point numbers is mapped onto one of the two
 - * A larger distance between two consecutive numbers results in a lower precision of representation

Formats

- ◆ Significand M and exponent E - signed quantities
- ◆ Exponent - usually a signed integer
- ◆ Significand - usually one of two:
 - * pure fraction, or
 - * a number in the range $[1, 2)$ (for $b=2$)
- ◆ Representing negative values - can be different
- ◆ Until 1980 - no standard - every computer system had its own representation method
 - * transporting programs/data between two different computers was very difficult
- ◆ **IEEE standard 754** is now used in most floating-point arithmetic units - details later
- ◆ Few computer systems use formats differing in partitioning of the n bits, representation of each part, or value of the base b

Significand Field

- ◆ Common case - signed-magnitude fraction
- ◆ Floating-point format - sign bit **S**, **e** bits of exponent **E**, **m** bits of unsigned fraction **M** ($m+e+1=n$)

S	Exponent E	Unsigned Significand M
----------	-------------------	-------------------------------

- ◆ Value of **(S,E,M)** : $F = (-1)^S \cdot M \cdot \beta^E$
 $((-1)^0 = 1 ; (-1)^1 = -1)$

- ◆ Maximal value - $M_{max} = 1 - ulp$
- ◆ **ulp** - Unit in the last position - weight of the least-significant bit of the fractional significand
- ◆ Usually - not always - $ulp = 2^{-m}$

The Base **b**

- ◆ **b** is restricted to 2^k ($k=1,2,\dots$) - simplifies decreasing significand and increasing exponent (and vice versa) at the same time
- ◆ Necessary when an arithmetic operation results in a significand larger than $M_{max} = 1 - ulp$ - significand is reduced and exponent increased - value remains unchanged
- ◆ Smallest increase in **E** is **1**

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

- ◆ M/b - a simple arithmetic shift right operation if **b** is an integral power of radix
- ◆ If $b=r=2$ - shifting significand to the right by a single position must be compensated by adding **1** to exponent

Example

- ◆ Result of an arithmetic operation - 01.10100×2^{100}
- significand larger than M_{\max}
- ◆ Significand reduced by shifting it one position to the right, exponent increased by 1
- ◆ New result - 0.11010×2^{101}
- ◆ If $b=2^k$ - changing exponent by 1 is equivalent to shifting significand by k positions
- ◆ Consequently - only k -position shifts are allowed
- ◆ If $b=4=2^2$
 $01.10100 \times 4^{010} = 0.01101 \times 4^{011}$

Normalized Form

- ◆ Floating point representation not unique -
 $0.11010 \times 2^{101} = 0.01101 \times 2^{110}$
- ◆ With $E=111$ - significand= 0.00110 - loss of a significant digit
- ◆ Preferred representation - one with no leading zeros - maximum number of significant digits - **normalized form**
- ◆ Simplifies comparing floating-point numbers - a larger exponent indicates a larger number; significands compared only for equal exponents
- ◆ For $b=2^k$ - significand normalized if there is a nonzero bit in the first k positions
- ◆ **Example:** Normalized form of $0.00000110 \times 16^{101}$ is $0.01100000 \times 16^{100}$

Range of Normalized Fractions

- ◆ Range of significand is smaller than $[0, 1 - \text{ulp}]$
- ◆ Smallest and largest allowable values are
- ◆ $M_{\min} = 1/b$; $M_{\max} = 1 - \text{ulp}$
- ◆ Range of normalized fractions does not include the value **zero** - a special representation is needed
- ◆ A possible representation for **zero** - $M=0$ and any exponent E
- ◆ $E=0$ is preferred - representation of zero in floating-point is identical to representation in fixed-point
 - * Execution of a test for **zero** instruction simplified

Representation of Exponents

- ◆ Most common representation - biased exponent
- ◆ $E = E^{\text{true}} + \text{bias}$ (**bias** - constant; E^{true} - the true value of the exponent represented in two's complement)
- ◆ Exponent field - e bits ; range: $-2^{e-1} \leq E^{\text{true}} \leq 2^{e-1} - 1$
- ◆ Bias usually selected as magnitude of most negative exponent 2^{e-1} $0 \leq E \leq 2^e - 1$
- ◆ Exponent represented in the **excess 2^{e-1}** method
- ◆ Advantages:
 - * When comparing two exponents (for add/subtract operations) - sign bits ignored; comparison like unsigned numbers
 - * Floating-points with S, E, M format are compared like binary integers in signed-magnitude representation
 - * Smallest representable number has the exponent **0**

Example: Excess 64

- ◆ $e=7$
- ◆ Range of exponents in two's complement representation is $-64 \leq E^{\text{true}} \leq 63$
- ◆ 1000000 and 0111111 represent -64 and 63
- ◆ When adding bias 64 , the true values -64 and 63 are represented by 0000000 and 1111111
- ◆ This is called: **excess 64** representation
- ◆ Excess 2^{e-1} representation can be obtained by
 - * Inverting sign bit of two's complement representation, or
 - * Letting the values 0 and 1 of the sign bit indicate negative and positive numbers, respectively

Range of Normalized Floating-Point Numbers

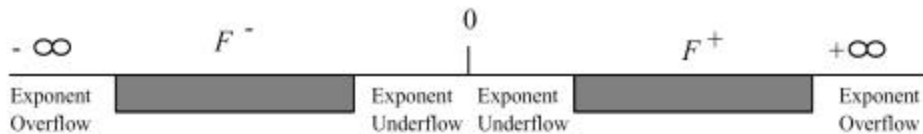
- ◆ Identical subranges for positive (F^+) and negative (F^-) numbers:

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

- * (E_{\min} , E_{\max} - smallest, largest exponent)
- ◆ An exponent larger than E_{\max} / smaller than E_{\min} must result in an exponent overflow/underflow indication
- ◆ Significand normalized - overflow reflected in exponent
- ◆ Ways of indicating overflow:
 - * Using a special representation of infinity as result
 - * stopping computation and interrupting processor
 - * setting result to largest representable number
- ◆ Indicating underflow:
 - * Representation of zero is used for result and an underflow flag is raised - computation can proceed, if appropriate, without interruption

Range of Floating Point Numbers

- ◆ Zero is not included in the range of either F^+ or F^-



Example - IBM 370

- ◆ Short floating-point format - 32 bits ; $b=16$

S - sign bit	E - 7 bits, excess 64 exponent	M - 24 bits, unsigned fractional significand
----------------	----------------------------------	--

$$F = (-1)^S \cdot M \cdot 16^{E-64}$$

- ◆ E_{min} , E_{max} represented by 0000000, 1111111 - value of -64, +63

- ◆ Significand - six hexadecimal digits

- ◆ Normalized significand satisfies -

$$M_{min} = 16^{-1} \leq M \leq M_{max} = 1 - 16^{-6} = 1 - 2^{-24}$$

- ◆ Consequently,

$$F_{max}^+ = (1 - 16^{-6}) \cdot 16^{63} \approx 7.23 \cdot 10^{75}$$

$$F_{min}^+ = (16^{-1}) \cdot 16^{-64} \approx 5.4 \cdot 10^{-79}$$

Numerical Example - IBM 370

◆ $(S, E, M) = (C1200000)_{16}$ in the short IBM format - first byte is $(11000001)_2$

- * Sign bit is $S=1$ - number is negative
- * Exponent is 41_{16} and, bias is $64_{10}=40_{16}$, $E^{\text{true}} = (41-40)_{16}=1$
- * $M=0.216$, hence $F = (-0.0010)_2 \cdot 16^1 = (-2)_{10}$.

◆ Resolution of representation - distance between two consecutive significands -

$$ulp = 16^{-6} = 2^{-24} \approx 0.6 \cdot 10^{-7}$$

- * Short format has approximately 7 significant decimal digits

◆ For higher precision use the long floating-point format

sign bit	7 bits - excess 64 exponent	56 bits - unsigned fractional significand
----------	-----------------------------	---

◆ Range roughly the same, but resolution:

$$ulp = 16^{-14} = 2^{-56} \approx 10^{-17}$$

- * 17 instead of 7 significant decimal digits

Floating-Point Formats of Three Machines

	IBM/370	DEC/VAX	Cyber 70
Word length (double)	32 (64) bits	32 (64) bits	60 bits
Significand+{hidden bit}	24 (56) bits	23+1 (55+1) bits	48 bits
Exponent	7 bits	8 bits	11 bits
Bias	64	128	1024
Base	16	2	2
Range of M	$\frac{1}{16} \leq M < 1$	$\frac{1}{2} \leq M < 1$	$1 \leq M < 2$
Representation of M	Signed-magnitude	Signed-magnitude	One's complement
Approximate range	$16^{63} \approx 7 \cdot 10^{75}$	$2^{127} \approx 1.9 \cdot 10^{38}$	$2^{1023} \approx 10^{307}$
Approximate resolution	$2^{-24} \approx 10^{-7} (10^{-17})$	$2^{-24} \approx 10^{-7} (10^{-17})$	$2^{-48} \approx 10^{-14}$

Hidden Bit

- ◆ A scheme to increase the number of bits in significand to increase precision
- ◆ For a **base of 2** the normalized significand will always have a leading **1** - can be eliminated, allowing inclusion of an extra bit
- ◆ The resolution becomes **ulp=2⁻²⁴** instead of **2⁻²³**
- ◆ The value of a floating-point number (**S, f, E**) in short DEC format is

$$(-1)^S 0.1f \cdot 2^{E-128}$$

- ◆ **f** - the pattern of **23** bits in significand field

Hidden Bit - representation of zero

- ◆ A **zero** significand field (**f=0**) represents the fraction **0.10₂=1/2**
- ◆ If **f=0** and **E=0** - with a hidden bit, this may represent the value **0.1 2⁰⁻¹²⁸ = 2⁻¹²⁹**
- ◆ The floating-point number **f=E=0** also represents **0** - a representation without a hidden bit
- ◆ To avoid double meaning - **E=0** reserved for representing **zero** - smallest exponent for **nonzero** numbers is **E=1**
- ◆ Smallest positive number in the **DEC/VAX** system -

$$F_{min}^+ = \frac{1}{2} 2^{1-128} = 2^{-128}$$

- ◆ Largest positive number -

$$F_{max}^+ = (1 - 2^{-24}) \cdot 2^{255-128} = (1 - 2^{-24}) \cdot 2^{127}$$

Floating-Point Operations

- ◆ Execution depends on format used for operands
- ◆ **Assumption:** Significands are normalized fractions in signed-magnitude representation ; exponents are biased

- ◆ Given two numbers

$$F_1 = (-1)^{S_1} \cdot M_1 \cdot \beta^{E_1 - \text{bias}} \quad ; \quad F_2 = (-1)^{S_2} \cdot M_2 \cdot \beta^{E_2 - \text{bias}}$$

- ◆ Calculate result of a basic arithmetic operation yielding

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

- ◆ Multiplication and division are simpler to follow than addition and subtraction

Floating-Point Multiplication

- ◆ Significands of two operands multiplied like fixed-point numbers - exponents are added - can be done in parallel
- ◆ Sign S_3 positive if signs S_1 and S_2 are equal - negative if not
- ◆ When adding two exponents
 $E_1 = E_1^{\text{True}} + \text{bias}$ and $E_2 = E_2^{\text{true}} + \text{bias}$:
bias must be subtracted once
- ◆ For $\text{bias} = 2^{e-1}$ (100...0 in binary) - subtracting bias is equivalent to adding bias - accomplished by complementing sign bit
- ◆ If resulting exponent E_3 is larger than E_{max} / smaller than E_{min} - overflow/underflow indication must be generated

Multiplication - postnormalization

- ◆ Multiplying significands M_1 and M_2 - M_3 must be normalized
- ◆ $1/b \leq M_1, M_2 < 1$ - product satisfies $1/b^2 \leq M_1 \times M_2 < 1$
- ◆ Significand M_3 may need to be shifted one position to the left
- ◆ Achieved by performing one base- b left shift operation - k base-2 shifts for $b=2^k$ - and reducing the exponent by 1
- ◆ This is called the **postnormalization** step
- ◆ After this step - exponent may be smaller than E_{min} - exponent underflow indication must be generated

ECE666/Koren Part. 4a.21

Copyright 2008 Koren

Floating-Point Division

- ◆ Significands divided - exponents subtracted - bias added to difference $E_1 - E_2$
- ◆ If resulting exponent out of range - overflow or underflow indication must be generated
- ◆ Resultant significand satisfies $1/b \leq M_1/M_2 < b$
- ◆ A single base- b shift right of significand + increase of 1 in exponent may be needed in postnormalization step - may lead to an overflow
- ◆ If divisor=0 - indication of **division by zero** generated - quotient set to $\pm \infty$
- ◆ If both divisor and dividend=0 - result undefined
 - in the **IEEE 754** standard represented by **NaN**
 - **not a number** - also representing uninitialized variables and the result of $0 \times \infty$

ECE666/Koren Part. 4a.22

Copyright 2008 Koren

Remainder in Floating-Point Division

- ◆ Fixed-point remainder - $R = X - QD$ (X , Q , D - dividend, quotient, divisor) - $|R| \leq |D|$ - generated by division algorithm (restoring or nonrestoring)
- ◆ Flp division - algorithm generates quotient but not remainder - $F1 \text{ REM } F2 = F1 - F2 \times \text{Int}(F1/F2)$
($\text{Int}(F1/F2)$ - quotient $F1/F2$ converted to integer)
- ◆ Conversion to integer - either truncation (removing fractional part) or rounding-to-nearest
- ◆ The IEEE standard uses the **round-to-nearest-even** mode - $|F1 \text{ REM } F2| \leq |F2| / 2$
- ◆ $\text{Int}(F1/F2)$ as large as $b^{E_{\max} - E_{\min}}$ - high complexity
- ◆ Floating-point remainder calculated separately - only when required - for example, in argument reduction for periodic functions like sine and cosine

Floating-Point Remainder - Cont.

- ◆ **Brute-force** - continue direct division algorithm for $E_1 - E_2$ steps
- ◆ **Problem** - $E_1 - E_2$ can be much greater than number of steps needed to generate m bits of quotient's significand - may take an arbitrary number of clock cycles
- ◆ **Solution** - calculate remainder in software
- ◆ **Alternative** - Define a **REM-step** operation - $X \text{ REM } F2$ - performs a limited number of divide steps (e.g., limited to number of divide steps required in a regular divide operation)
- ◆ Initial $X = F1$, then X = remainder of previous **REM-step** operation
- ◆ **REM-step** repeated until **remainder** $\leq F2/2$

Addition and Subtraction

- ◆ Exponents of both operands must be equal before adding or subtracting significands
- ◆ When $E_1=E_2$ - b^{E_1} can be factored out and significands M_1 and M_2 can be added
- ◆ Significands aligned by shifting the significand of the smaller operand $|E_1-E_2|$ base- b positions to the right, increasing its exponent, until exponents are equal
- ◆ $E_1 \geq E_2$ - $F_1 \pm F_2 = \left((-1)^{S_1} \cdot M_1 \pm (-1)^{S_2} \cdot M_2 \cdot \beta^{-(E_1-E_2)} \right) \cdot \beta^{E_1 - bias}$
- ◆ Exponent of larger number not decreased - this will result in a significand larger than 1 - a larger significand adder required

Addition/Subtraction - postnormalization

- ◆ **Addition** - resultant significand M (sum of two aligned significands) is in range $1/b \leq M < 2$
- ◆ If $M > 1$ - a postnormalization step - shifting significand to the right to yield M_3 and increasing exponent by one - is required (an exponent overflow may occur)
- ◆ **Subtraction** - Resultant significand M is in range $0 \leq |M| < 1$ - postnormalization step - shifting significand to left and decreasing exponent - is required if $M < 1/b$ (an exponent underflow may occur)
- ◆ In extreme cases, the postnormalization step may require a shift left operation over all bits in significand, yielding a **zero** result

Example

- ◆ $F_1 = (0.100000)_{16} \times 16^3$; $F_2 = (0.FFFFFFF)_{16} \times 16^2$
- ◆ Short IBM format ; calculate $F_1 - F_2$

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

- ◆ Significand of smaller number (F_2) is shifted to the right - least-significant digit lost
- ◆ Shift is time consuming - result is wrong

Example - Cont.

- ◆ Correct result (with "unlimited" number of significand digits)

F_1	0.	1	0	0	0	0	0	0	·	16^3
F_2 aligned	0.	0	F	F	F	F	F	F	·	16^3
$F_1 - F_2$	0.	0	0	0	0	0	0	1	·	16^3
Postnormalization	0.	1	0	0	0	0	0	0	·	16^{-3}

- ◆ Error (also called loss of significance) is
 - ◆ $0.1 \times 16^{-2} - 0.1 \times 16^{-3} = 0.F \times 16^{-3}$
- ◆ Solution to problem - **guard digits** - additional digits to the right of the significand to hold shifted-out digits
- ◆ In example - a single (hexadecimal) guard digit is sufficient

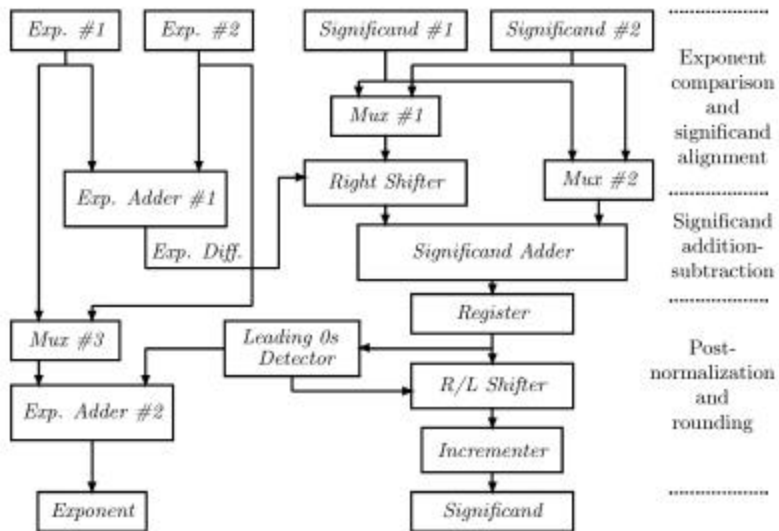
Steps in Addition/Subtraction of Floating-Point Numbers

- ◆ **Step 1:** Calculate difference d of the two exponents - $d = |E1 - E2|$
- ◆ **Step 2:** Shift significand of smaller number by d base- b positions to the right
- ◆ **Step 3:** Add aligned significands and set exponent of result to exponent of larger operand
- ◆ **Step 4:** Normalize resultant significand and adjust exponent if necessary
- ◆ **Step 5:** Round resultant significand and adjust exponent if necessary

ECE666/Koren Part. 4a.29

Copyright 2008 Koren

Circuitry for Addition/Subtraction



ECE666/Koren Part. 4a.30

Copyright 2008 Koren

Shifters

- ◆ 1st shifter - right (alignment) shifts only ; 2nd shifter - right or left (postnormalization) shifts ; both perform large shifts (# of significand digits)
- ◆ **Combinatorial shifter** - generate all possible shifted patterns - only one at output according to control bits
 - * Such shifters capable of circular shifts (rotates) - known as **barrel shifters**
 - * Shift registers require a large and variable number of clock cycles, thus combinatorial shifters commonly used
- ◆ If implemented as a single level array - each input bit is directly connected to **m** (or more) output lines - conceptually simple design
- ◆ For **m=53** (number of significand bits in IEEE double-precision format) - large number of connections (and large electrical load) - bad solution

ECE666/Koren Part. 4a.31

Copyright 2008 Koren

Two levels Barrel Shifters

- * first level shifts bits by 0, 1, 2 or 3 bit positions
 - * second level shift bits by multiples of 4 (0,4,8,...,52)
 - * shifts between 0 and 53 can be performed
 - ◆ **Radix-4 shifter**
 - ◆ **16 bits**
-
- * 1st level - each bit has 4 destinations ; 2nd level - each bit has 14 destinations - unbalanced
 - ◆ **Radix-8 shifter** - 1st level shifts 0 to 7 bit positions ; 2nd level shifts by multiples of 8 (0,8,16,24,...,48)
 - * 1st level - each bit has 8 destinations ; 2nd level - each bit has 7 destinations

ECE666/Koren Part. 4a.32

Copyright 2008 Koren

Choice of Floating-Point Representation

- ◆ **IEEE standard 754** commonly used - important to understand implications of a particular format
- ◆ Given n - total number of bits - determine
 - * m - length of significand field
 - * e - length of exponent field ($m+e+1=n$)
 - * b - value of exponent base
- ◆ **Representation error** - error made when using a finite-length floating-point format to represent a high-precision real number
- ◆ x - a real number ; $FI(x)$ - its machine representation
- ◆ **Goal** when selecting format - small representation error
- ◆ Error can be measured in several ways

ECE666/Koren Part. 4a.33

Copyright 2008 Koren

Measuring Representation Error

- ◆ Every real number x has two consecutive representations F_1 and F_2 satisfying $F_1 \leq x \leq F_2$
- ◆ $FI(x)$ can be set to either F_1 or F_2
- ◆ $FI(x) - x$ - absolute representation error
- ◆ $d(x) = (FI(x) - x) / x$ - relative representation error
- ◆ If $F_1 = Mb^E$ then $F_2 = (M+ulp)b^E$
- ◆ Maximum absolute error = half distance between F_1 and $F_2 = ulp \cdot b^E$ - increases as exponent increases



ECE666/Koren Part. 4a.34

Copyright 2008 Koren

Measure of Representation Accuracy

- ◆ **MRRE** - maximum relative representation error - upper bound of $d(x)$

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

- ◆ **MRRE** increases with exponent base β - decreases with **ulp** (or number of significand bits m)
- ◆ Good measure if operands uniformly distributed
- ◆ In practice - larger significands less likely to occur
- ◆ First digit of a decimal floating-point operand will most likely be a **1**; **2** is the second most likely
- ◆ Operands follow the density function

$$\frac{1}{M \ln \beta}; \quad 1/\beta \leq M \leq 1$$

ECE666/Koren Part. 4a.35

Copyright 2008 Koren

Different Accuracy Measure

- ◆ **ARRE** - average relative representation error
- ◆ Absolute error varies between **0** and $1/2 \text{ulp} \beta^E$
- ◆ Average absolute error is $1/4 \text{ulp} \beta^E$
- ◆ Relative error is $1/4 \text{ulp}/M$

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

ECE666/Koren Part. 4a.36

Copyright 2008 Koren

Range of Representation

- ◆ The **range** of the positive floating-point numbers - $b^{E_{max}}$ - must be considered when selecting a floating-point format
- ◆ For a large range - increase **b** and/or number of exponent bits **e**
- ◆ Increasing **b** increases representation error
- ◆ Increasing **e** decreases **m** and increases **ulp** - higher representation error
- ◆ Trade-off between range and representation error

Range - Accuracy Trade-off

β	e	m	Range	MRRE	ARRE
2	9	22	$2^{2^9-1} = 2^{255}$	$0.5 \cdot 2^{-22} \cdot 2 = 2^{-22}$	$0.180 \cdot 2^{-21}$
4	8	23	$4^{2^7-1} = 2^{2^8-2} = 2^{254}$	$0.5 \cdot 2^{-23} \cdot 4 = 2^{-22}$	$0.135 \cdot 2^{-21}$
16	7	24	$16^{2^6-1} = 2^{2^8-4} = 2^{252}$	$0.5 \cdot 2^{-24} \cdot 16 = 2^{-21}$	$0.169 \cdot 2^{-21}$

- ◆ If several floating-point representations have same range - select smallest **MRRE** or **ARRE**
- ◆ If several representations have same **MRRE** (or **ARRE**) - select the largest range
- ◆ **Example**: 32-bit word - $m+e=31$ - all three representations have about the same range
- ◆ Using **MRRE** as measure - $b=16$ inferior to other two
- ◆ Using **ARRE** as measure - $b=4$ is best
- ◆ $b=2$ + hidden bit reduces **MRRE** and **ARRE** by a factor of 2 - the smallest representation error

Execution Time of Floating-Point Operations

- ◆ One more consideration when selecting a format
- ◆ Two time-consuming steps - aligning of significands before add/subtract operations ; postnormalization in any floating-point operation
- ◆ **Observation** - larger b - higher probability of equal exponents in add/subtract operations - no alignment ; lower probability that a postnormalization step needed
- ◆ No postnormalization in
59.4% of cases for $b=2$;
82.4% for $b=16$
- ◆ This is of limited practical significance when a barrel shifter is used

Alignment shift	$\beta = 16$	$\beta = 2$
0	47.3%	32.6%
1	26.0%	12.1%
≥ 2	26.7%	55.3%